INDUSTRIAL CONTROL COMMUNICATIONS, INC.

EtherNet/IP Server Driver Manual

October 30, 2014

© 2014 Industrial Control Communications, Inc.

TABLE OF CONTENTS

1 EtherNet/IP Server 2

1.1 Overview 2

1.2 Supported Class 3 Services 2

1.3 Server Settings 3

1.3.1 Class 1 I/O Messaging Settings 4

1.4 Service and Access Format Reference 4

1.4.1 Class 1 (I/O) Connection Access 4

1.4.2 Class 3 (Explicit Messaging) Via Data Table Read/Write 4

1.4.2.1 Internal Database Access Tag Format 4

1.4.2.2 Mitsubishi MELSEC Bypass Access Tag Format 5

1.4.3 Class 3 (Explicit Messaging) Via Typed Read/Write 6

1.5 Application Examples 7

1.5.1 ControlLogix Examples: Setup 7

1.5.2 ControlLogix Example: Class 1 (I/O) Messaging 8

1.5.3 ControlLogix Example: Read Database Data 11

1.5.4 ControlLogix Example: Multiple MSG Instructions 15

1.5.5 ControlLogix Example: Reading and Writing 15

1.5.6 ControlLogix Example: Read MELSEC Device Data 17

1.5.7 SLC-5/05 Example: Read Database Data 17

1.5.8 SLC-5/05 Example: Multiple MSG Instructions 20

1.5.9 SLC-5/05 Example: Reading and Writing 21

1 EtherNet/IP Server

1.1 Overview

This driver supports the EtherNet/IP server protocol, including the CSP variant. Both class 1 (I/O) and class 3 (explicit messaging) connections are supported.

In this section, the terms "O→T" and "T→O" are EtherNet/IP specification nomenclature which stands for "Originator to Target" and "Target to Originator", respectively. Typically, the originator is the client and the target is the server.

Ethernet-enabled Allen-Bradley legacy PLCs (such as the PLC-5/E, SLC-5/05 and MicroLogix platforms) use a protocol called CSP (Client Server Protocol) to communicate via Ethernet. The variant of CSP used by these PLCs is also known as "PCCC" (Programmable Controller Communication Commands) and "AB Ethernet". The driver supports CSP via the "Typed Read" and "Typed Write" (also known as "PLC5 read" and "PLC5 write") services for direct connectivity to these PLCs.

Some notes of interest are:

- The driver supports the EtherNet/IP protocol (release 1.0), administered by the Open DeviceNet Vendor Association (ODVA).
- This product has been self-tested by ICC, Inc. and found to comply with ODVA EtherNet/IP Conformance Test Software Version A-6.
- I/O connection sizes for assembly instances 100 and 150 are limited by the driver only in that they must lie entirely within the internal database boundaries.
- The identity object's product type code is 12 (Communications Adapter).
- The driver supports unconnected messages (UCMM), and up to 16 simultaneous class 1 (I/O) or class 3 (explicit) connections.
- Class 1 and class 3 connections operate independently of each other.
- Class 1 I/O connections support both multicast and point-to-point (unicast) when producing data in the T→O direction.
- If a class 1 point-to-point connection is established in the (T→O) direction, no further class 1 server connections can be established.
- If a class 1 connection's consuming half (O→T) times out, then the producing half (T→O) will also time-out and will stop producing.
- The driver will trigger a timeout event whenever a class 1 or class 3 connection timeout or socket-level error occurs.
- Special database bypass / on-demand access for Mitsubishi MELSEC PLC applications.

1.2 Supported Class 3 Services

The driver supports the following class 3 (explicit messaging) services:

Data Table Read

Reads data associated with a tag name.

Data Table Write

Writes data associated with a tag name.

PLC5 Read / Typed Read

Reads a block of data associated with a file number and offset.

PLC5 Write / Typed Write

Writes a block of data associated with a file number and offset.

Class 3 access requires no additional driver configuration: predefined tag names and file numbers exist by default when the driver is deployed.

1.3 Server Settings

Device Name

The device name is used for identification of a device on the EtherNet/IP network. This string is accessible as the "product name" attribute of the identity object. Enter a string between 1 and 32 characters in length.

Run/Idle Flag Behavior

EtherNet/IP clients (such as PLCs) have the option of adding a 32-bit "run/idle" header to all class 1 (I/O) data packets sent to devices. Bit 0 of this header is called the "run/idle flag" by the EtherNet/IP specification, and is intended to signify when the client is in a "running" state or an "idle" state. A running state (run/idle flag = Run) is indicated whenever the client is performing its normal processing (e.g. scanning its ladder logic). An idle state (run/idle flag = Idle) is indicated otherwise. For example, Allen Bradley ControlLogix PLCs will set their run/idle flag to Idle whenever their processor keyswitch is placed in the "PROG" position, presumably in preparation to receive a new application program from RSLogix 5000.

The behavior of EtherNet/IP devices when they receive I/O data from a controller with the run/idle flag set to Idle is not defined in the EtherNet/IP specification. The driver allows the option of two different behavioral responses when a run/idle flag = Idle condition is received, depending on the state of the *Invoke Timeout When Run/Idle Flag = Idle* checkbox.

- If the checkbox is cleared (default setting), then the driver will maintain the last I/O data values received from the client. For example, if a device mapped to the database was being commanded to run prior to the run/idle flag being set to Idle by the client, then it will continue to run.
- If the checkbox is checked, then the driver will trigger a timeout event for user-defined timeout processing. This setting allows the user to determine any "failsafe" behavior they may desire.

1.3.1 Class 1 I/O Messaging Settings

When the EtherNet/IP server driver is added to the configuration, class 1 communication capability is added by default, and cannot be disabled. The driver continually listens for incoming class 1 connection requests.

Produced Data Start Address

Defines the starting database address from which class 1 produced data will be retrieved by the driver and sent to the client. The size of the produced data is determined by the client upon initial connection establishment.

Consumed Data Start Address

Defines the starting database address to which class 1 consumed data will be stored by the driver when received from the client. The size of the consumed data is determined by the client upon initial connection establishment.

1.4 Service and Access Format Reference

Depending on the connection class and specific services to be used by the client, conventions for specific pre-defined assembly instances, tag name formats, etc. must be followed. These conventions are dictated by the server driver, and must be followed by the client in order to properly access the desired data.

1.4.1 Class 1 (I/O) Connection Access

The structure of I/O messaging consumed and produced assembly membership lists is entirely user-configurable by appropriate management of data residing in the internal database. Clients may access the class 1 endpoint by opening a connection to assembly instances 100 and 150. For an application example, refer to section 1.5.2.

1.4.2 Class 3 (Explicit Messaging) Via Data Table Read/Write

Data table read and write services provide a direct method of accessing the internal database, as well as an on-demand bypass mechanism to directly access data residing on other networked devices by reference to “tag names”. Tags are read via the EtherNet/IP “data table read” service, and written via the EtherNet/IP “data table write” service. Note that an underscore (“_”) is used to separate fields in the tag names.

To read data, the client must reference a starting “source element” and the “number of elements” to read. Similarly, to write data, the client must reference a starting “destination element” and the “number of elements” to write. The “number of elements” can be any quantity from 1 to the maximum allowable length, while the “source element” and “destination element” must be tag names constructed according to the naming conventions shown in this section.

1.4.2.1 Internal Database Access Tag Format

Any given internal database address can be accessed with its own unique tag name. Tag names must be generated according to the following structure:

[target prefix]_[data type]_[address]

Where
[target prefix] is a 2-character sequence “DB” to indicate database access.

[data type] is a 1-character field that specifies the data type of the values in the response. Enter “b” or “B” to indicate byte (8-bit). Enter “w” or “W” to indicate word (16-bit). Enter “d” or “D” to indicate double word (32-bit). Note that the storage/source element on the requester must also be of the same data size.

[address] is a 1- to 4-character field (“0”, “1”, “2”, etc.) corresponding to the referenced database address. Specify the value in decimal format.

Examples
Access bytes starting at database address 0.....DB_b_0
Access words starting at database address 2048 .....DB_w_2048
Access double words starting at database address 4092 .....DB_d_4092

For application examples, refer to sections 1.5.3, 1.5.4 and 1.5.5.

1.4.2.2 Mitsubishi MELSEC Bypass Access Tag Format

Any device data on a Mitsubishi MELSEC server (iQ PLC, etc.) can be accessed with its own unique tag name, assuming the knowledge of some MELSEC client driver connection object information. Note that the MELSEC bypass access mechanism does not require the use of any predefined MELSEC client driver service objects: MELSEC client driver requests are generated on-demand in direct response to corresponding tag accesses. Tag names must be generated according to the following structure:

[target prefix]_[cnxn name]_[device code]_[starting point][_bit flag]

Where
[target prefix] is a 2-character sequence “MC” to indicate MELSEC bypass request.

[cnxn name] is a 1- to 16-character field specifying the MELSEC connection object name.

[device code] is a field of up to 4 characters in length that corresponds to the device code on the MELSEC device. This device code must be specified in hexadecimal radix format (do not prepend the code with the “0x” characters typically associated with hexadecimal notation). Enter either a 2-character hexadecimal value for 3E frames or a 4-character hexadecimal value for 1E frames. Device codes are not limited only to known device codes: any device code supported by the targeted server may be specified. If specifying a MELSEC word device, the storage/source element on the requester must be a 16-bit data type. If specifying a MELSEC bit device, the storage/source element on the requester must be an 8-bit data type. If the device code is a bit device but is unknown the MELSEC client driver, the bit flag must be set (see bit flag below). Device codes are not case-sensitive (i.e. “C3”=”c3”).

[starting point] is a 1- to 8-character decimal value corresponding to the starting point of the device code.

[_bit flag] is an optional 2-character field "_b" which indicates whether or not an unknown device code targets a bit device. If the device code is unknown to the MELSEC client driver and this field is omitted, then it is assumed that the unknown device code targets a word device. This field is only relevant when the targeted device code is unknown to the MELSEC client driver: if the device code is known, then the presence or absence of this field is ignored.

Examples

Access Request	Tag
Using connection object "Q04", target Data Register (0xA8) starting at device point 0	MC_Q04_A8_0
Using connection object "Q04", target Input Relay (0x9C) starting at device point 1000	MC_Q04_9C_1000
Using connection object "QJ71E71", target unknown word device (0xFF) starting at device point 125	MC_ QJ71E71_FF_125
Using connection object "QJ71E71", target unknown bit device (0xFE) starting at device point 1250	MC_ QJ71E71_FE_1250_b

Notes

- *When accessing a bit device on a MELSEC server, each bit is actually encoded as an entire byte by the server. If the targeted bit is 0, then the resultant byte is 0; else the resultant byte is non-zero.*
- *Please refer to the MELSEC server equipment user's manual for limitations on the number of points that can be processed in a single transaction.*

For an application example, refer to section 1.5.6.

1.4.3 Class 3 (Explicit Messaging) Via Typed Read/Write

Data can be read from and written to the internal database via CSP by reference to an integer "file/section number" and an "offset/element" within that file. Reading is performed via the CSP "PLC5 Read" (DF1 protocol "Typed Read") service, and writing is performed via the CSP "PLC5 Write" (DF1 protocol "Typed Write") service. Both Logical Binary Addressing and Logical ASCII Addressing are supported. Only the Integer file type 'N' is supported, and data is accessible only as 16-bit values.

The formula to calculate which database address is targeted is provided in Equation 1.

$$target\ address = (file\ number\ -10) \times 100 + offset \qquad \textbf{Equation 1}$$

In Equation 1, "target address" $\in$[0…4094], "file number" $\in$[10…50] (which means N10…N50), and "offset" is restricted only by the limitations of the programming software (but is a value of 4094 max). Table 1 provides various example combinations of file/section numbers and offsets/elements which can be used to access database addresses. Note that multiple different combinations of file/section numbers and offsets/elements exist which result in the same database address being accessed.

Table 1: CSP (Typed Read/Write) Target Database Address Examples

Logical ASCII Addressing	File/Section Number	Offset/Element	Start Target Address
N10:2	N10	2	2
N12:62	N12	62	262
N11:162	N11	162	262
N27:98	N27	98	1798
N20:798	N20	798	1798
N50:95	N50	95	4095
N10:4094	N10	4094	4094

For application examples, refer to sections 1.5.7, 1.5.8 and 1.5.9.

1.5 Application Examples

The following sections demonstrate typical examples of how to use class 1 and class 3 services to transfer data between the driver and Allen-Bradley –Logix and legacy PLCs.

1.5.1 ControlLogix Examples: Setup

This section will demonstrate how to initially setup a ControlLogix PLC (such as a 1756-L61) coupled with a 1756-ENBT/A communication interface (adjust this procedure according to your specific equipment). Later sections will provide specific read/write examples using this configuration with I/O or explicit messaging.

1) Run RSLogix 5000, and create a new configuration.
2) To add a 1756-ENBT/A to your I/O configuration, first switch to offline mode.
3) Right click on the I/O Configuration node in the controller organizer view and choose “New Module…”
4) The “Select Module” window will open.
5) Under “Communications”, select “1756-ENBT/A”, and click OK. Refer to Figure 1.
6) The “New Module” window will open. Refer to Figure 2.
7) Assign the Ethernet module a name (we will use “EIP”) and an IP address, deselect “Open Module Properties”, and click OK.
8) Download the configuration.

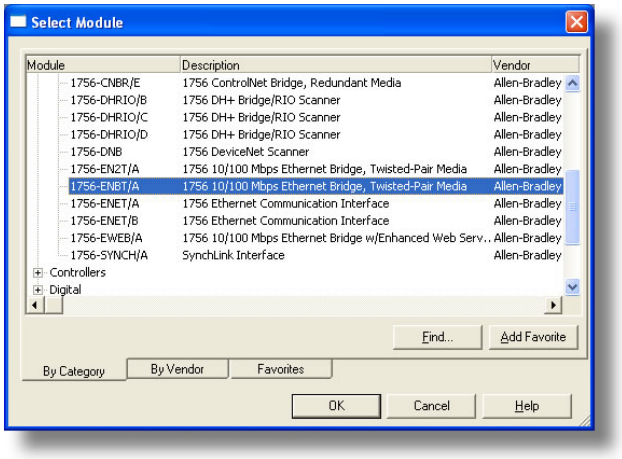

Figure 1: Adding a New 1756-ENBT/A Module

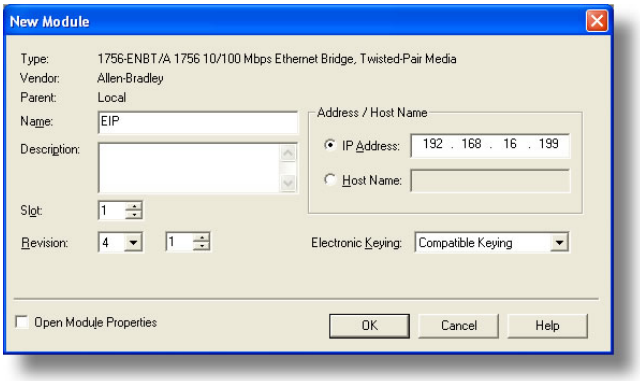

Figure 2: Configuring the New Module

9) Switch to online mode. Right click on the 1756-ENBT/A module in the I/O Configuration and choose “Properties”.

10) Select the Port Configuration tab from the Module Properties dialog box.

11) Confirm that the IP Address, Subnet Mask and Gateway Address fields are configured correctly. The IP Address must match the IP Address entered when the new module was first created. Refer to Figure 3.

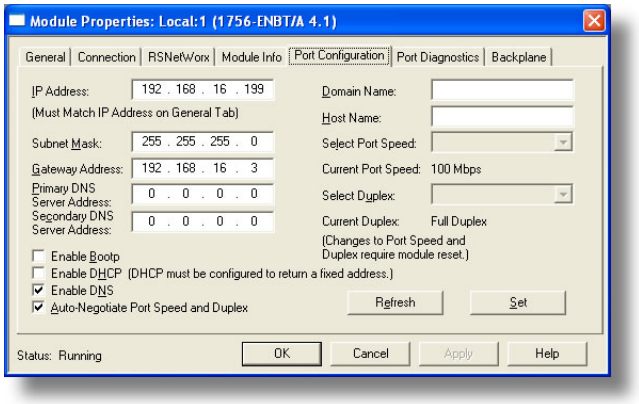

Figure 3: Confirming the Module's Properties

12) Apply any changes to the settings using the “Set” button.

13) You should now be able to confirm that the 1756-ENBT/A module is configured properly by (for example) opening the module’s web interface in a web browser.

1.5.2 ControlLogix Example: Class 1 (I/O) Messaging

This section will demonstrate how to setup and use an EtherNet/IP I/O connection via vendor-specific assembly instances 100 & 150. EtherNet/IP I/O messaging allows the internal database to be directly mapped into tags in the ControlLogix PLC. Once an I/O connection is established, it is automatically synchronized at an interval defined by the Requested Packet Interval (RPI).

1) Switch to offline mode.

2) Right click on the 1756-ENBT/A node under the I/O Configuration in the controller organizer view and choose “New Module…”

3) Choose “Generic Ethernet Module” in the Select Module dialog box and click “OK”. Refer to Figure 4.

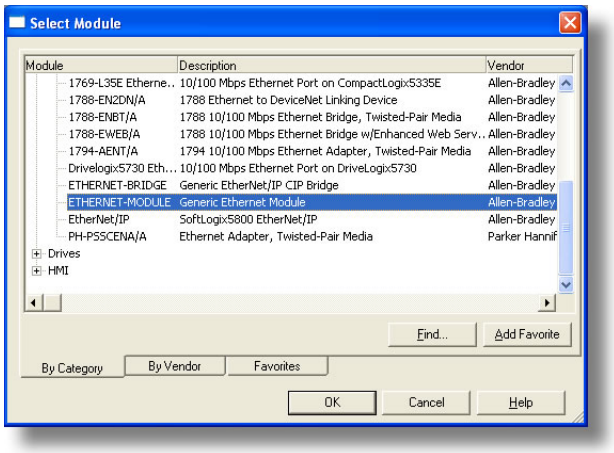

Figure 4: Adding a New Generic Ethernet Module

4) The module properties dialog box will open (refer to Figure 5). Enter a Name and Description which will allow easy identification of the device on the network (the tags created in RSLogix 5000 will be derived from this Name). Because the data type of the internal database is entirely subject to interpretation, an appropriate “Comm Format” must be chosen which makes the transferred data meaningful to the application. For this specific example, the “Comm Format” is set to “Data-SINT”. Enter the IP address of the targeted device.

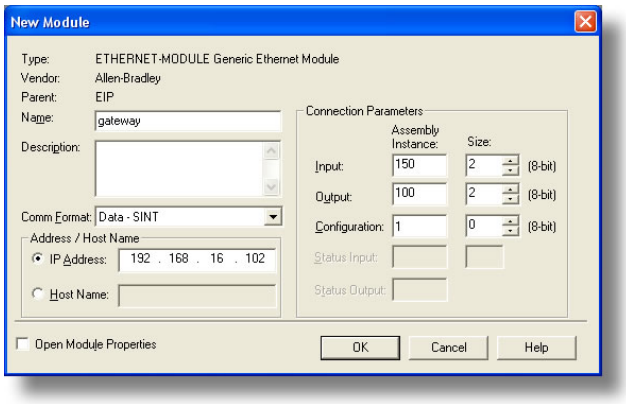

Figure 5: Module Properties

In the “Connection Parameters” portion of the dialog box, enter the following information:

Input: The Input Assembly is the collection of monitor data that is produced by the driver and is received as an input to the PLC. The starting address for produced data is specified by the *Produced Data Start Address* (refer to section 1.3.1). The Input Assembly Instance must be set to 150 when connecting to the vendor-specific I/O endpoint, and the size must be set to the number of elements (each of “Comm Format” type) that we wish to receive.

Notice that the “number of bits” specified to the right of the “size” selection box changes according to the selected “Comm Format” (the “number of bits” label shown in Figure 5 is “8-bit”). As the driver will reject any connection requests that attempt to access data past the upper boundary of the database, Equation 2 can be used to determine if the overall requested size is valid. If the result of Equation 2 is less than or equal to the maximum database address, then the driver will accept the connection request.

$$size\ x\ (\frac{number\ of\ bits}{8}) + Produced\ Data\ Start\ Address$$ **Equation 2**

Output: The Output Assembly is the collection of command & configuration data that is sent as an output from the PLC and consumed by the driver. The starting address for consumed data is specified by the *Consumed Data Start Address* (refer to section 1.3.1). The Output Assembly Instance must be set to 100 when connecting to the vendor-specific I/O endpoint, and the size must be set to the number of elements (each of “Comm Format” type) that we wish to send to the driver. In a similar manner as the input assembly previously discussed, Equation 3 can be used to determine if the overall requested size is valid. If the result of Equation 3 is less than or equal to the maximum database address, then the driver will accept the connection request.

$$size\ x\ (\frac{number\ of\ bits}{8}) + Consumed\ Data\ Start\ Address$$ **Equation 3**

Configuration: The Configuration Assembly Instance is unused, and its instance number and size are therefore irrelevant (you can just enter “1” and “0”, respectively).

When done, click “OK”.

5) You should now see the new module (named “ETHERNET-MODULE gateway”) in the 1756-ENBT/A branch under the I/O Configuration in the controller organizer view. Right click on this new module, choose “Properties”, and select the Connection tab. Refer to Figure 6.

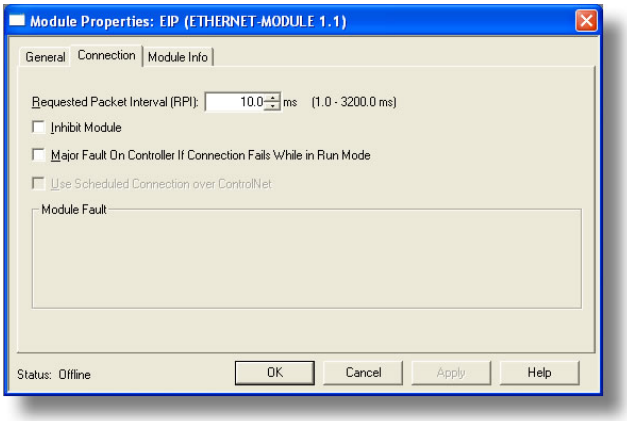

Figure 6: Module Properties Connection Tab

Confirm the setting of the Requested Packet Interval (RPI). The RPI defines the amount of time (in milliseconds) between data exchanges across an I/O connection. The smallest RPI supported by the driver is 10ms.

When done, click “OK”.

6) After adding the I/O Module to the configuration, the full I/O Configuration tree should appear similar to Figure 7.

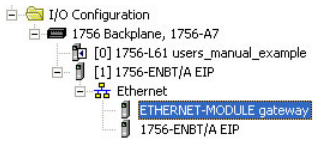

Figure 7: I/O Configuration Tree

7) Switch to online mode and download the project to the PLC. Verify that the newly-added device is available and operating correctly by observing any indications shown on the device's icon. When the device's icon is selected, its status and any available error messages will be displayed in the area below the project tree. Refer to Figure 8. Also confirm that the device's "Network Status" LED should be solid green, indicating an "online/connected" state.

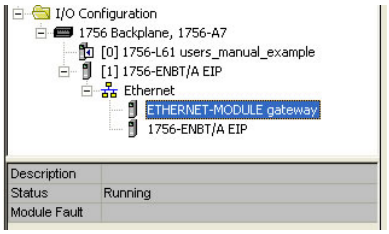

Figure 8: Online Module Status

8) By double-clicking "Controller Tags" in the project tree, it is possible to view the newly-added tags. Refer to Figure 9. The gateway:C configuration tag is unused, the gateway:I tag allows viewing of the input data, and the gateway:O tag allows modification of the output data. These tags will be synchronized with the driver at whatever rate was established for the module's RPI. If desired, we can directly interact with these tags in order to read and write data in the database.

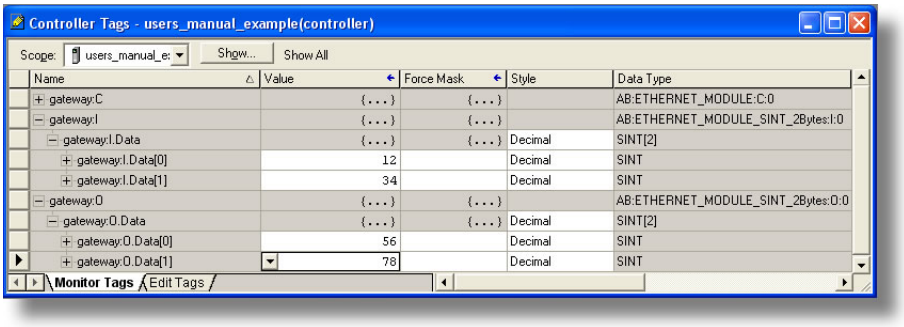

Figure 9: Controller Tags for I/O Access

1.5.3 ControlLogix Example: Read Database Data

This example program will show how to continuously read a block of data from the internal database with a single MSG instruction. This action is performed via the CIP Data Table Read message type. Only one read request is outstanding at any given time.

1) Create new Tags.

a) Double click "Controller Tags" in the controller organizer view.

b) The "Controller Tags" window appears. Refer to Figure 10.

c) Select the "Edit Tags" tab at the bottom.

d) Create a new tag by entering "connection" in the first blank Name field, and change its Data Type to "MESSAGE". This tag will contain configuration information for the MSG instruction.

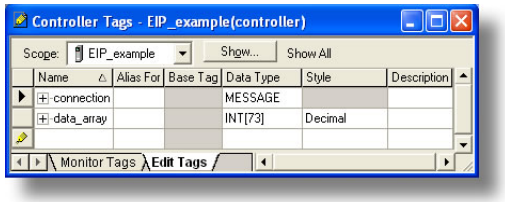

Figure 10: Create New Tags

e) Select the “Monitor Tags” tab. Expand the “connection” tag by clicking on the “+” sign next to the tag name. Scroll down to the connection.UnconnectedTimeout field and change its value from the default 30000000 (30s in 1uS increments) to 1000000 (1s). This value determines how long to wait before timing out and retransmitting a connection request if a connection failure occurs. Refer to Figure 11.

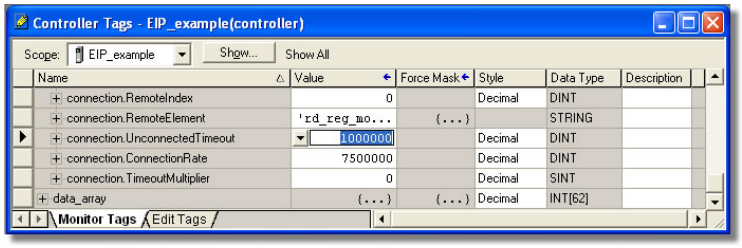

Figure 11: Reduce the UnconnectedTimeout Value

f) Collapse the “connection” tag again by clicking on the “-“ sign next to the tag name.

g) Select the “Edit Tags” tab again. Create another new tag by entering “data_array” in the next blank Name field, and change its Data Type by typing in “INT[73]” in the Data Type field. This tag is an array of INTs that will be able to hold up to 73 16-bit values from the database. Always make sure that the destination tag size is large enough to hold all elements to be read.

2) Add a MSG instruction to the main program.

a) Double click “MainRoutine” under Tasks …MainTask …MainProgram in the controller organizer view.

b) Right click on the first ladder logic rung in the MainRoutine window and select “Add Ladder Element...”

c) The “Add Ladder Element” window appears.

d) Select the “MSG” instruction in the Input/Output folder. Refer to Figure 12.

e) Click OK.

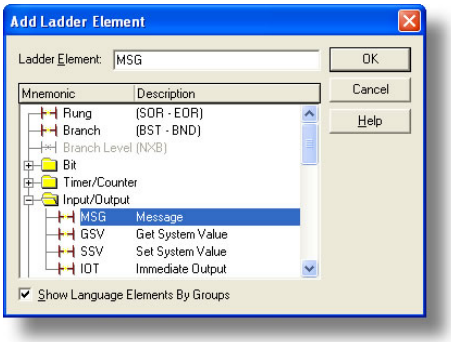

Figure 12: Adding a MSG Instruction

3) Add an XIO element to the main program.

a) Right click on the ladder logic rung containing the MSG instruction in the MainRoutine window and select “Add Ladder Element...” again.

b) The “Add Ladder Element” window appears.

c) Select the “XIO” element in the Bit folder. Refer to Figure 13.

d) Click OK.

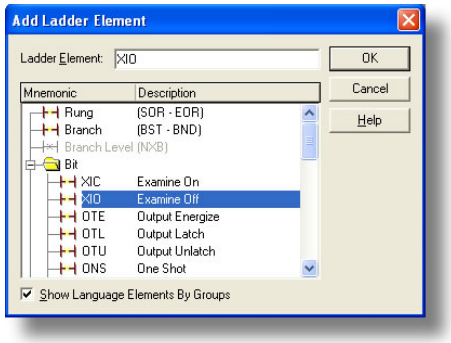

Figure 13: Adding an XIO Element

4) Configure the MSG instruction.

a) Edit the “Message Control” field on the MSG instruction to use the previously-created “connection” tag. Refer to Figure 14.

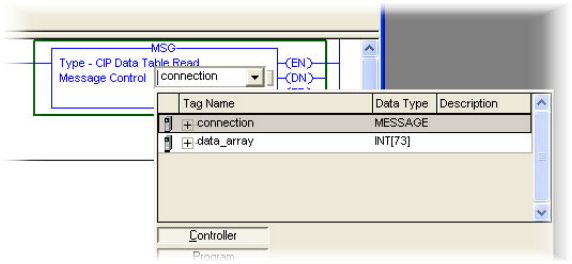

Figure 14: MSG Instruction Tag Assignment

b) Click the message configuration button (“…”) in the MSG instruction. The “Message Configuration” window will open. Refer to Figure 15.

Figure 15: MSG Instruction Configuration

c) “Configuration” tab settings:

i) Change the “Message Type” to “CIP Data Table Read”.

ii) In the "Source Element” field, enter the read tag you wish to access (refer to section 1.4.2.1). In this example, we will be reading a total of 21 16-bit words beginning at DB_w_2050 (database address 2050).

iii) Enter the Number Of Elements to read. In this example, we will read 21 16-bit words.

iv) For the Destination Element, either directly type in “data_array[50]”, or select element #50 in the data_array tag via the drop-down box. The destination could be any offset in the data_array tag, as long as the offset plus the Number Of Elements (21) does not exceed the tag’s defined size (73).

d) “Communication” tab settings (refer to Figure 16):

i) Enter the Path to the device. A typical path is formatted as “*Local_ENB, 2, target_IP_address*”, where:

- *Local_ENB* is the name of the 1756-ENBx module in the local chassis (we named ours “EIP” in section 1.5.1),
- *2* is the Ethernet port of the 1756-ENBx module in the local chassis, and

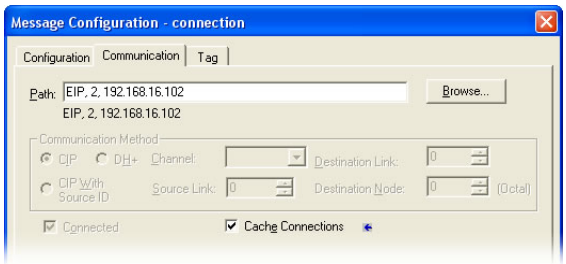

Figure 16: Setting the Communication Path

- *target_IP_address* is the IP address of the target node.

In our example, this path would be entered as “EIP, 2, 192.168.16.102”.

ii) If “Cache Connections” is enabled (checked), the connection remains open after transmission. If disabled (unchecked), the connection is opened before and closed after every transmission. For efficiency, it is recommended to enable “Cache Connections”.

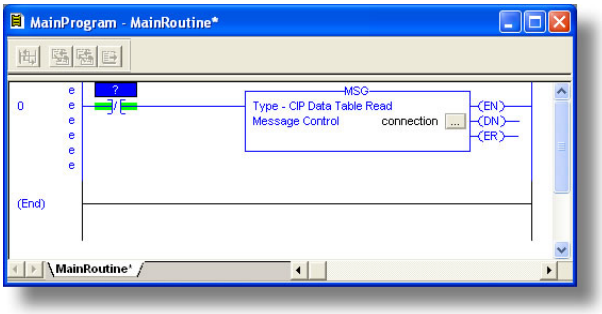

Figure 17: MainRoutine

e) Click “OK” to close the MSG Configuration dialog. At this stage, MainRoutine should look like Figure 17.

5) Assign a tag to the XIO element.

Double-click on the XIO element located to the left of the MSG block. In the drop-down box, double-click on the “connection.EN” field. Refer to Figure 18. This configuration causes the MSG instruction to automatically retrigger itself when it completes. While this is acceptable for the purposes of this example, it can produce high network utilization. In actual practice, it may be desirable to incorporate additional logic elements to allow triggering the MSG instruction at a specific rate or under specific conditions.

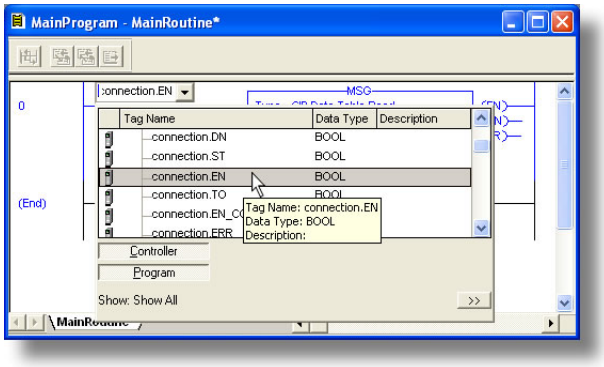

Figure 18: Configure XIO Element

6) The program is now complete. Refer to Figure 19.

7) Save, download and run the program.

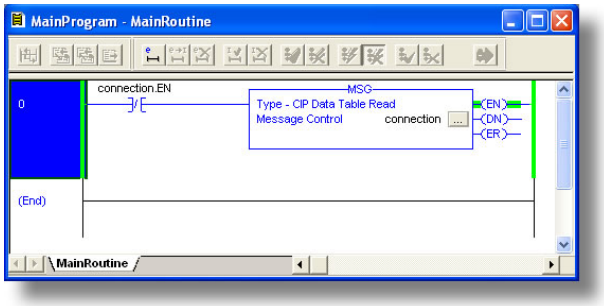

Figure 19: Complete Program

a) To view the values of the data being read from the database, double-click “Controller Tags” in the controller organizer view.

b) Select the “Monitor Tags” tab.

c) Expand the data_array tag. Refer to Figure 20.

d) 21 16-bit values starting at database address #2050 are being continuously read from the internal database and placed in the 21 sequential offsets of data_array starting at the 50th offset (data_array[50]). In Figure 20, we can see that data_array[50] has a value of 11860, data_array[58] has a value of 3558, etc.

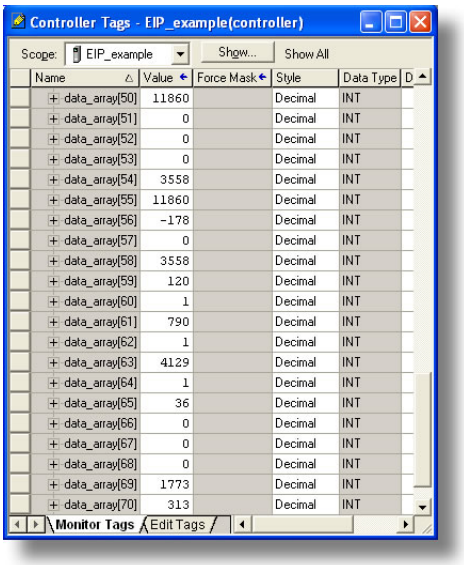

Figure 20: Viewing the Values

1.5.4 ControlLogix Example: Multiple MSG Instructions

At times, reading from different groups of data may be necessary. To accomplish this task, multiple MSG instructions will need to be implemented in the PLC program.

The configuration and execution for implementing multiple MSG instructions is in general identical to that required for implementing just one MSG instruction. Each MSG instruction will require its own message controller tag. In the case of read MSG instructions, more than one instruction may use the same Destination Element tag, but the storage locations must not overlap. Figure 21 shows an example of two MSG instructions, each accessing different read tags. It is evident from this logic that “rd_connection” and “rd_connection2” are the two independent message controller tags created for these instructions.

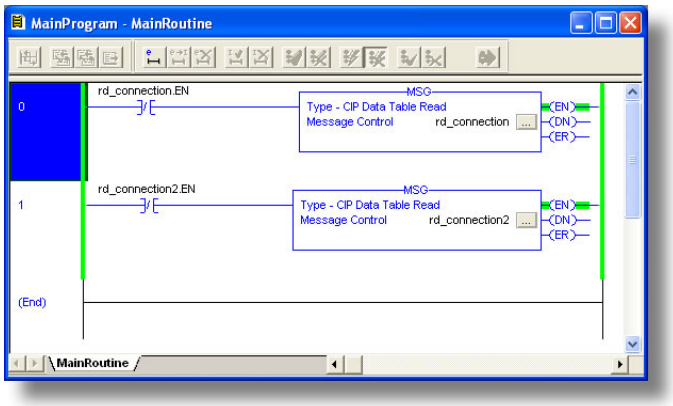

Figure 21: Reading Via Multiple MSG Instructions

1.5.5 ControlLogix Example: Reading and Writing

Often times, applications may need to both read data from and write data to the database. At a minimum, this will require two MSG instructions and two message controller tags. Figure 22

shows an example of three MSG instructions, one for reading and two for writing. The only item of note that differentiates this example from the multiple-read example in section 1.5.4 is the addition of the en_xx_wr XIC elements. The reason for the addition of these elements is that while reading from a remote device is often continuously performed (monitoring), data is typically written to the remote device only when necessary (i.e. when the value to write has changed). This conserves network bandwidth on the target device. The en_xx_wr elements in this example, therefore, would typically be replaced in an actual application program by user-provided logic that controls the conditions under which write operations would be performed.

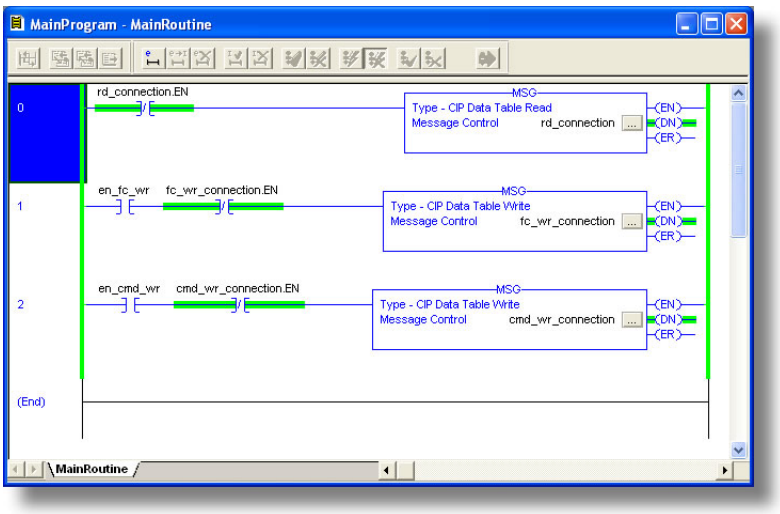

Figure 22: Reading and Writing via MSG Instructions

Figure 23 shows the configuration details of the example fc_wr_connection MSG instruction. Note that the chosen "Message Type" is "CIP Data Table Write", and that this instruction will be writing 21 16-bit values.

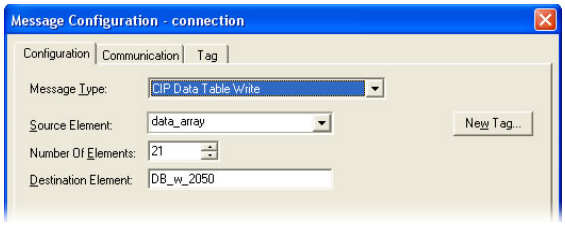

Figure 23: MSG Configuration for Writing

Note that when writing data via explicit messaging, use caution to ensure that the commanded addresses are not also simultaneously being commanded in the background via I/O (class 1) messaging. Indeterminate behavior can occur if MSG instructions and background I/O data transfers are both writing to the same addresses. In other words, if the I/O messaging example procedure detailed in section 1.5.2 has already been implemented, and the same program is now being modified to implement explicit messaging, then it is recommended to inhibit the target I/O module by selecting the "Inhibit Module" checkbox in the Connection tab of the Module Properties dialog.

1.5.6 ControlLogix Example: Read MELSEC Device Data

This example program will demonstrate bypass access to device data on a MELSEC server. The configuration and execution for implementing a MSG instruction is in general identical to the database access example in section 1.5.3. There are, however, two discrepancies to note:

1. The *Source Element* must take the form of a MELSEC tag name structure as outlined in section 1.4.2.2. In this example, the "cnxn_name" field is "Q04" (as derived from the MELSEC client connection object's name), the device code is a8 (data register), and the starting point is 0. Refer to Figure 24.
2. The data type of the *Destination Element* depends on the device code in use. If the device code targets a word device, then the *Destination Element* must be specified as type INT. If the device code targets a bit device, however, then the *Destination Element* must be specified as type SINT.

Figure 24: MELSEC Bypass MSG Instruction Configuration

This example of directly reading MELSEC device data via explicit messaging can easily be extended to accesses via multiple MSG instructions and to writing data by following the basic examples detailed in sections 1.5.4 and 1.5.5, respectively.

1.5.7 SLC-5/05 Example: Read Database Data

This example program will show how to continuously read a block of data from the internal database with a single MSG instruction. This action is performed via the Typed Read (a.k.a. "PLC5 Read") message type. Only one read request is outstanding at any given time. Note that the steps for the MicroLogix and PLC5E may vary slightly, but in general are similar.

1) Run RSLogix 500, and create a new configuration.

2) Create a control and a data file.

a) Right click Data Files and select New… The "Create Data File" dialog box appears (refer to Figure 25).

b) To create a control file, enter a file number (e.g. 20), set the type to "Integer", enter a descriptive name (e.g. "CONTROL"), and enter a number of elements (e.g. 100). Click OK to create the file. The control file is used to store configuration information pertaining to the functionality of the MSG instruction which will perform the data read.

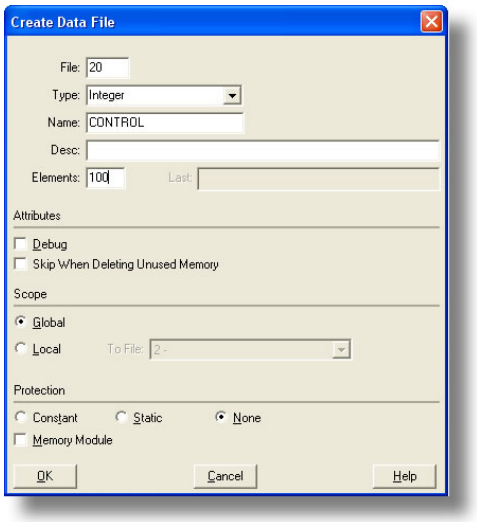

Figure 25: Creating a Control File

c) Follow the same procedure to create a data file. This file will be used to store the incoming data read from the internal database. Enter a file number (e.g. 18), set the type to "Integer", enter a descriptive name (e.g. "DATA"), and enter a number of elements (e.g. 200). Refer to Figure 26. Click OK to create the file.

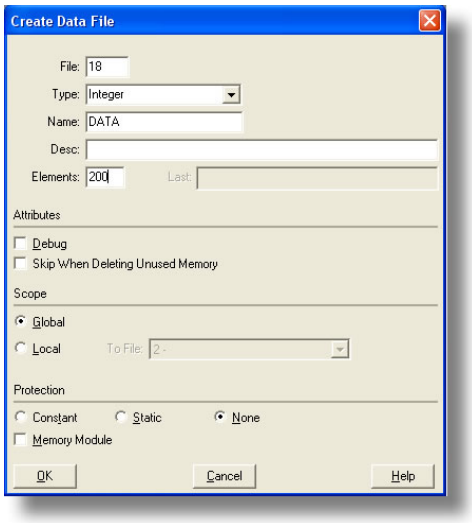

Figure 26: Creating a Data File

3) Add a MSG instruction to the program.

a) If not already visible, double-click "LAD2" under Project…Program Files in the controller organizer view to bring up the ladder logic program.

b) Right click on the default rung number on the left-hand side of the LAD2 window and select "Insert Rung".

c) Right click on the rung number of the new editable rung and select "Append Instruction".

d) Select the "MSG" instruction from the "Input/Output" classification, and then click OK. Refer to Figure 27.

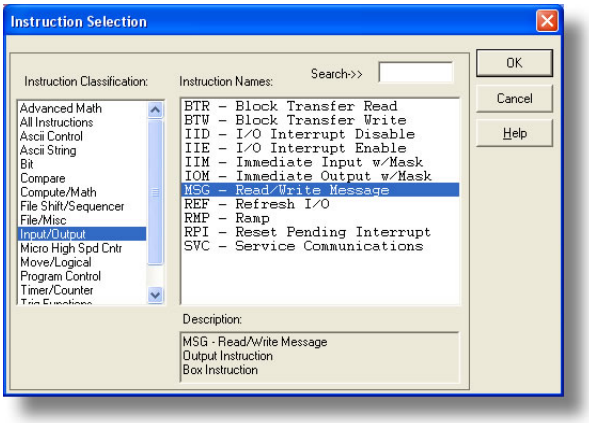

Figure 27: MSG Instruction Selection

4) Add an XIO element to the program.

a) Right click on the rung number of the rung currently being edited and select "Append Instruction" again.

b) Select the "XIO" instruction from the "Bit" classification, and then click OK. Refer to Figure 28.

5) Configure the MSG instruction.

a) Set the "Read/Write" field to "Read", "Target Device" field to "PLC5", "Local/Remote" field to "Local", and "Control Block" to "N20:0".

b) Upon hitting the <ENTER> key while in the "Control Block" entry box, the MSG Properties dialog box should appear (or it can be opened by clicking on the "Setup Screen" button at the bottom of the MSG instruction). Refer to Figure 29.

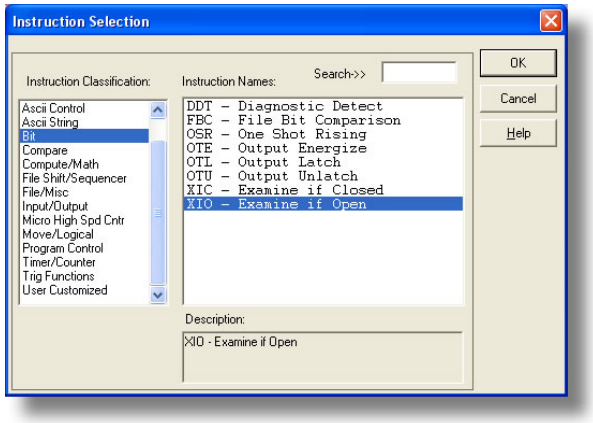

Figure 28: XIO Instruction Selection

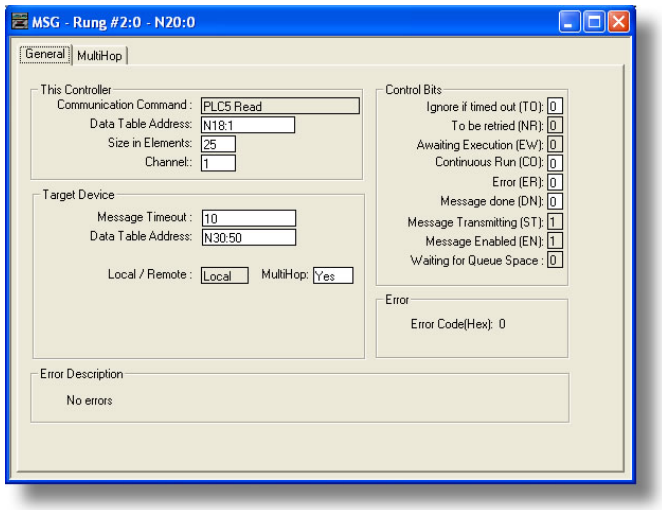

Figure 29: MSG Configuration, "General" Tab

c) In this example, we will be reading a total of 25 16-bit word values from the database beginning at N30:50 (address 2050). To configure this, under “This Controller” set the “Data Table Address” field to N18:1, set the “Size in Elements field” to 25, and set the “Channel” field to 1 (Ethernet).

d) Under “Target Device”, set the “Data Table Address” field to N30:50 (starting target address=2050) and set the “MultiHop” field to Yes to cause the “MultiHop” tab to appear.

e) Under the “MultiHop” tab settings, set the “To Address” in the first row to the device’s IP address, and the “To Address” in the second row to 0. Refer to Figure 30.

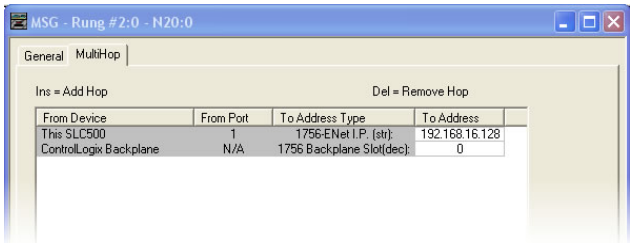

Figure 30: MSG Configuration, "MultiHop" Tab

f) Close the dialog box. At this point, the program should appear as shown in Figure 31.

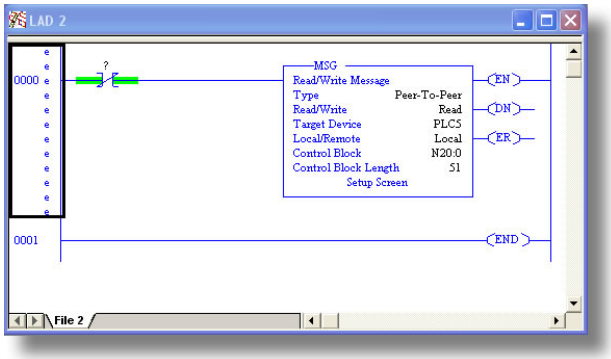

Figure 31: PLC Program after MSG Instruction Configuration

6) Assign a tag to the XIO element.

a) Double-click on the XIO element located to the left of the MSG block. Type in N20:0/15 (MSG instruction's enable bit). This configuration causes the MSG instruction to automatically retrigger itself when it completes. While this is acceptable for the purposes of this example, it can produce high network utilization. In actual practice, it may be desirable to incorporate additional logic elements to allow triggering the MSG instruction at a specific rate or under specific conditions.

7) The program is now complete. Refer to Figure 32.

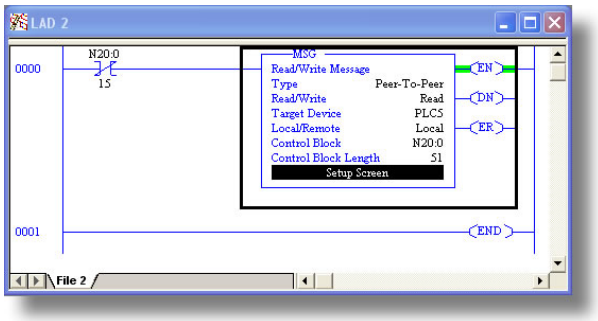

Figure 32: Completed PLC Program

8) Save, download, and run the program.

a) To view the 16-bit data values being read from the internal database, double-click the data file N18 under "Data Files" in the controller organizer view. A total of 25 16-bit word values starting at address #2050 are being continuously read from the database and placed in the 25 sequential offsets of N18 starting at N18:1. Refer to Figure 33. We can see that N18:9 (database address 2068) has a value of 2525, N18:12 (database address 2074) has a value of 610, etc.

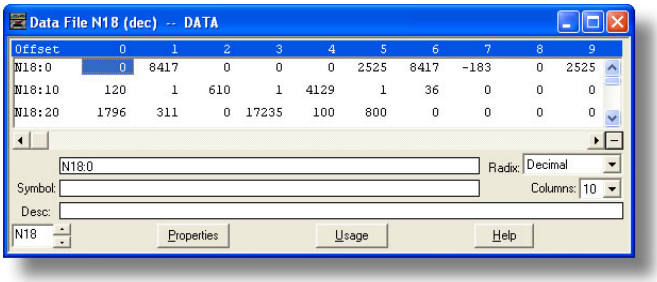

Figure 33: Monitoring the Data Being Read from the Database

1.5.8 SLC-5/05 Example: Multiple MSG Instructions

At times, reading from different non-contiguous address of the database may be necessary. To accomplish this task efficiently, multiple MSG instructions can be implemented in the PLC program.

The configuration and execution for implementing multiple MSG instructions is in general identical to that required for implementing just one MSG instruction. Each MSG instruction will

require its own message control file. In the case of read MSG instructions, more than one instruction may use the same data file to store the received 16-bit word values, but the storage locations must not overlap. Figure 34 shows an example of two MSG instructions, each accessing different target integer files. It is evident from this logic that N20 and N21 are the two independent message control files created for these instructions.

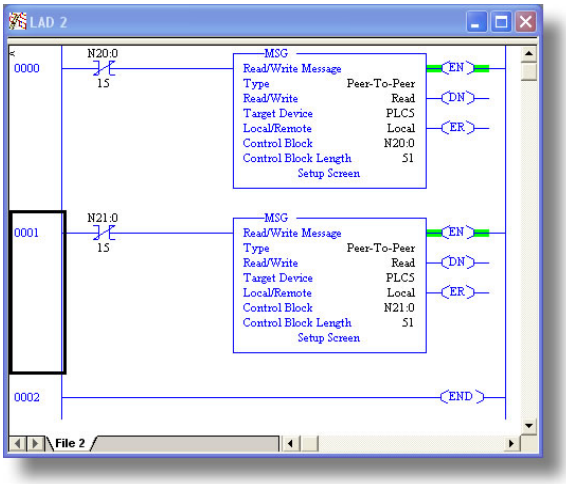

Figure 34: Multiple MSG Instructions

1.5.9 SLC-5/05 Example: Reading and Writing

Often times, applications may need to both read data from and write data to the internal database. At a minimum, this will require two MSG instructions and two message control files. Figure 35 shows an example of two MSG instructions, one for reading and one for writing. Note that the "Read/Write" field of each MSG instruction is set according to their function.

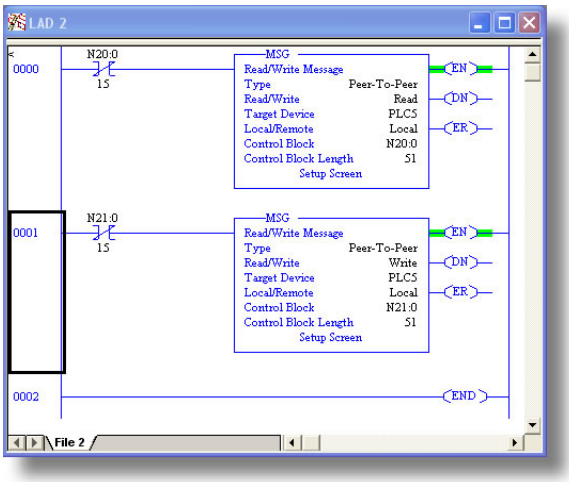

Figure 35: Reading and Writing via MSG Instructions

Figure 36 shows the configuration details of the "write" MSG instruction. Note that this instruction will only be writing to one 16-bit word value (N27:98 = database address 1798). The source Data Table Address in this case is N18:30.

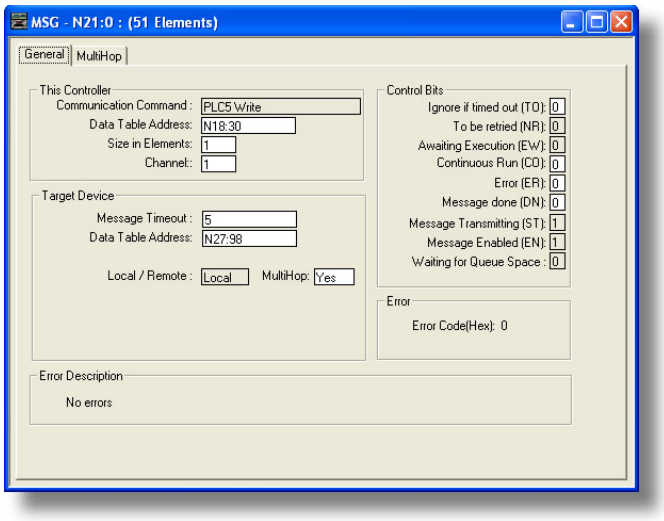

Figure 36: MSG Configuration for Writing

ICC

INDUSTRIAL CONTROL COMMUNICATIONS, INC.

1600 Aspen Commons, Suite 210
Middleton, WI USA 53562-4720
Tel: [608] 831-1255 Fax: [608] 831-2045

http://www.iccdesigns.com

Printed in U.S.A